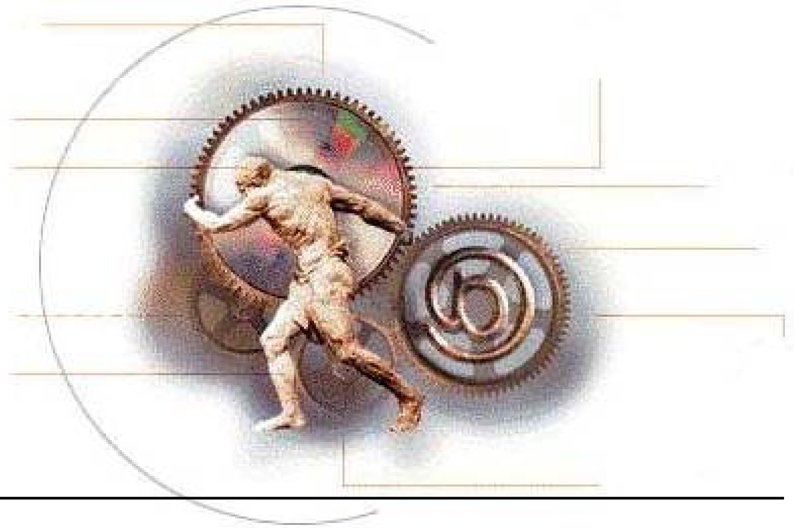


Course 2

遞迴

Recursion

國立聯合大學 資訊管理學系 陳士杰老師

Outlines

- 本章重點
  - Def.與種類
  - Recursion與Non-recursion的比較
  - 設計方式
  - 遞迴演算法則的複雜度分析
    - 數學解法 (Mathematics-based method)
    - 替代法 (Substitution method)
    - 遞迴樹法 (Recursion tree method)
    - 支配定理法 (Master theorem method)

Recursion Algorithm

- 一般來說，有兩種方式可以撰寫具有重覆執行(Repetitive)特性的演算法:
  - **Iteration (迴圈)**
  - **Recursion (遞迴)**
- **Def**: algorithm 中含有**self-calling** (自我呼叫)敘述存在。

- 遞迴的種類:
  - **直接遞迴 (Direct Recursion):**
    - 函式或程序直接呼叫本身時稱之。

```
void Function2(void)
{
........
Function2( );
........
}
```

  - **間接遞迴 (Indirect Recursion):**
    - 函式或程序先呼叫另外的函式，再從另外函式呼叫原來的函式稱之。

```
void Function1(void)
{
........
Function2( );
........
}
```

```
void Function2(void)
{
........
Function1( );
........
}
```

- **尾端遞迴 (Tail Recursion):**
  - 屬於直接遞迴的特例

程式結束的前一行

```
void Function2(void)
{
………
………
Function2( );
}
```

- 建議：用**非遞迴**方式會較有效率
  - 即: 改用**迴圈** (while…, repeat…until)
  - ∵遞迴要花費額外的處理 (如: stack的push, pop,…)

遞迴演算法則的設計

1. 找出問題的**終止條件 (即：base case)**.
2. 找出問題本身的**遞迴關係 (即：general case)**.

- 遞迴的架構:

```
Procedure 遞迴副程式名(參數)
{
 if (base case)
   return(結果); ……//達到終止條件時結束遞迴, 需要時回傳結果
 else
   general case; ……//利用general case執行遞迴呼叫, 需要時加上return
}
```

階乘 (Factorial; n!)

$$\text{Factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1 & \text{if } n > 0 \end{cases}$$

We can define

終止條件

$$\text{Factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \times (\text{Factorial}(n-1)) & \text{if } n > 0 \end{cases}$$

遞迴關係

Recursive Factorial Algorithm

inputs: n is the number being raised factorially

outputs: n! is returned

```
Procedure Factorial(int n)
begin
    if (n = 0)
        return 1;
    else
        return (n× Factorial(n-1));
end
```

Write a program in C++

```
int Factorial(int n)
{
 if (n==0)
     return (1);
 else
     return (n*Factorial(n-1));
}
```

Iterative Factorial Algorithm

inputs: n is the number being raised factorially

outputs: n! is returned

```
void Factorial(int n)
{
    factN = 1;
    for (i=1, i ≤ n, i++)
        factN = factN * i;
    return factN;
}
```

費氏數 (Fibonacci Number)

- Ex:

n	0	1	2	3	4	5	6	7	8	9	10
F_n	0	1	1	2	3	5	8	13	21	34	55

- 觀念:

$F_0 + F_1 \Rightarrow F_2$

$F_1 + F_2 \Rightarrow F_3$

$F_2 + F_3 \Rightarrow F_4$

$F_3 + F_4 \Rightarrow F_5$

$\Rightarrow F_a + F_b \Rightarrow F_c$

◆Recursive Algorithm Definition

終止條件

$$F_n = \begin{cases} 0, & if \quad n = 0 \\ 1, & if \quad n = 1 \\ F_{n-1} + F_{n-2}, & if \quad n \geq 2 \end{cases}$$

遞迴關係

Recursive Fibonacci Algorithm

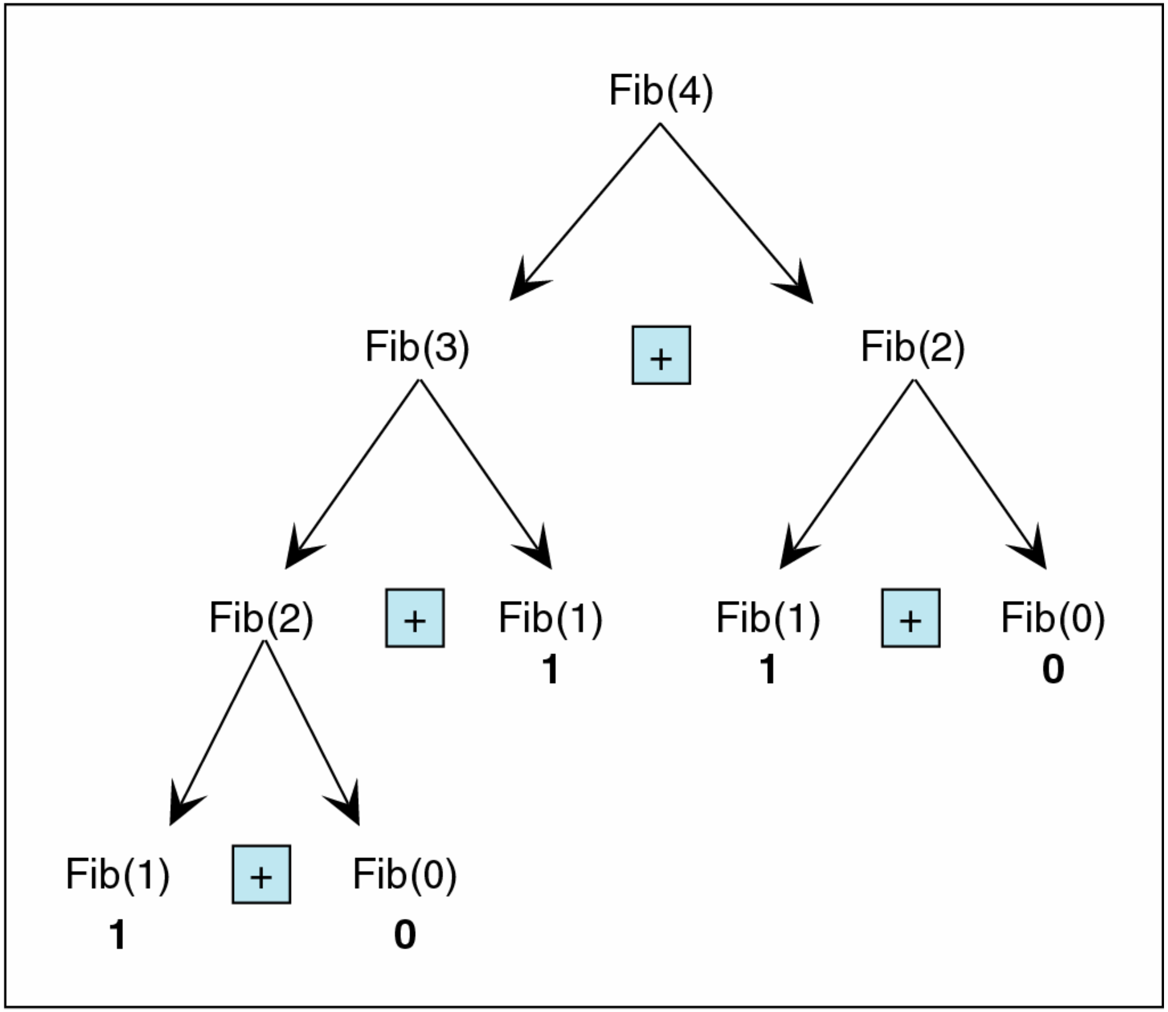
inputs: num identified the ordinal of the Fibonacci number

outputs: returns the nth Fibonacci number

```
void Fib(int num)
{
    if (num is 0 OR num is 1)
        return num;
    else
        return (Fib(num-1) + Fib(num-2));
}
```

- **Based on recursive function, 求Fib (4) 共呼叫此函數幾次? (含Fib(4))**

➪ **Ans: 9次**



(b) Fib(4)

Iterative Fibonacci Number Algorithm

inputs: num identified the ordinal of the Fibonacci number

outputs: returns the nth Fibonacci number

```
void Fib(int num)
{
    if (num is 0 OR num is 1)
        return num;
    else {
        Fa = 0, Fb = 1;
        for(i = 2, i<=num, i++)
            Fc = Fa + Fb;
            Fa = Fb;
            Fb = Fc;
        end for;
        return Fc;
    };
}
```

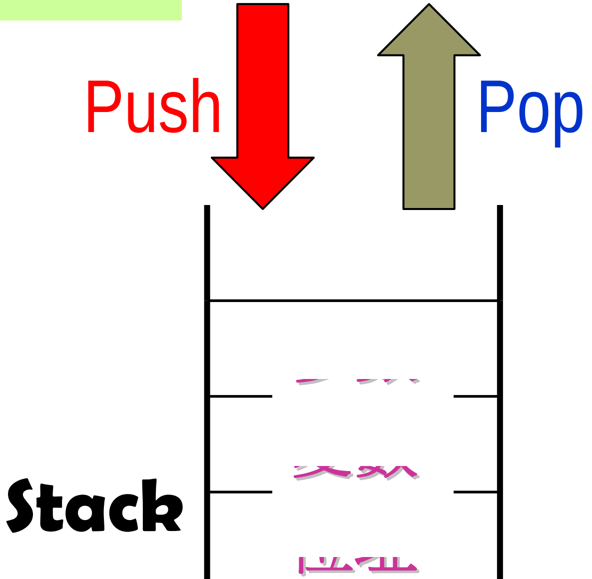
C++ Program:

```
int Fib(int n)
{
  if (n <= 1)
      return n;
  else {
        int Fa=0, Fb=1, Fc, i;
        for(i=2; i<=n; i++)
           {
             Fc = Fa + Fb;
             Fa = Fb;
             Fb = Fc;
           };
        return Fc;
       };
}
```

How Recursion Works

```
void Function1(void)
{
……..
Function2( );
……..
}
```

```
void Function2(void)
{
……..
……..
End;
}
```



① 要**保存Function1當時執行的狀況**，即Push下列資料到Stack中。

- **參數值 (Parameter)**
- **區域/暫存變數值 (Local Variable)**
- **返回位址 (Return Address)**

② 要做**控制權轉移** (Jump to Function2)

③ Recursion動作結束時，要**Pop Stack**，以取出參數、區域/暫存變數值及返回位址，then goto “Return Address”。

⇨Push, Jump, Pop皆**耗時**，∴**效率差**

◆ Recursion與Non-recursion的程式可以**互相改寫**!!

■ Recursion 與 Non-recursion 的比較

	Recursion	Non-recursion	
優	◆	◆	缺
	◆	◆	
	◆	◆	
	◆	◆	
缺	◆	◆	優
	◆	◆	

Note:

可自行回答下列問題，若有一個回答為“no”，則你不應使用遞迴來設計演算法:

1. 演算法所處理的問題或是資料結構本身合乎遞迴的特性嗎 **(Is the algorithm or data structure naturally suited to recursion)?**
2. 若用了遞迴是否可使結果更小及更易了解 **(Is the recursive solution shorter and more understandable)?**
3. 這個遞迴結果的執行是在可接受的執行時間和空間限制嗎 **(Does the recursive solution run in acceptable time and space limits)?**

遞迴演算法則的複雜度分析

- 遞迴演算法的分析比iterative algorithm的分析要來得困難。
- 分析步驟:
  - 我們找出遞迴演算法的**遞迴方程式 T(n) (recurrence equation,** 或通稱爲**Time function, Complexity function**亦可**)** 來表達該演算法的執行次數。
  - 接著，解這個遞迴方程式來求出該演算法的**時間複雜度**。

遞迴演算法的遞迴方程式

- 以 "Factorial" 爲例，透過對遞迴關係與終止條件的分析，我們可以得知Factorial遞迴演算法的遞迴方程式 (時間函數) 如下:

$$T(n) = T(n-1) + 1$$

- **每執行一次遞迴呼叫後，接著可能還需要之 "遞迴呼叫的執行次數"**
  - 根據遞迴演算法中的General Case得知
- 在此，爲第n次的遞迴呼叫執行後，尚需n-1次的遞迴呼叫工作

- **在每一次遞迴呼叫中，所需花費之 "非遞迴工作的執行次數"**
  - 根據不同的問題，會求得不同的非遞迴工作之執行次數
- 在此，爲第 n 次的遞迴呼叫執行中，需要執行1次的非遞迴乘法工作

(也可寫成 c 表示某一常數)

- 以 "Fibonacci" 爲例，透過對遞迴關係與終止條件的分析，我們可以得知Fibonacci遞迴演算法的遞迴方程式 (時間函數) 如下:

$$T(n) = \underline{T(n-1) + T(n-2)} + 1$$

- 每執行一次遞迴呼叫後，接著可能還需要之 "遞迴呼叫的執行次數"
- 在此，爲第n次的遞迴呼叫執行後，尚需兩個不同的遞迴呼叫工作

- 在每一次遞迴呼叫中，需花費之 "非遞迴工作的執行次數"
- 在此，爲第 n 次的遞迴呼叫執行中，需要執行1次的非遞迴加法工作

(也可寫成 c 表示某一常數)

- 通常在討論遞迴演算法時，我們常會一起將這些演算法的遞迴方程式列出。因此，本單元假設遞迴方程式已給定，主要議題則設定在如何解遞迴方程式。
- 遞迴方程式的解法與使用時機:

解法	使用時機
數學解法 (Mathematics-based Method)	逼不得已時 (求解過程較煩瑣, 但幾乎任何遞迴方程式皆可求解)
遞迴樹法 (Recursion-tree Method)	母問題由多個子問題所構成時
支配理論 (Master Theorem Method)	遞迴方程式為特定型式時
替代法 (Substitution Method)	已知Answer時 (要用猜的, 且結論有時較難匯整)

數學解法 (Mathematics-based Method)

- 即課本附錄B.3的代入法 (Substitution)
  - 意義不同於後面介紹的Substitution Method (替代法)
- 直接將遞迴方程式以遞迴的觀念由最末項往前求解，然後整理出答案。

範例 1

- 請利用數學解法找出下列遞迴方程式 (時間函數) 的時間複雜度。

$$T(n) = T(n-1) + n$$

$$T(1) = 1$$

- 遞迴呼叫的執行次數
- 非遞迴工作的執行次數

Sol:

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= (T(n-2) + (n-1)) + n = T(n-2) + (n-1) + n \\ &= (T(n-3) + (n-2)) + (n-1) + n = T(n-3) + (n-2) + (n-1) + n \\ &\ \ \ldots \\ &= T(1) + 2 + 3 + \ldots + (n-2) + (n-1) + n \\ &= 1 + 2 + 3 + \ldots + (n-2) + (n-1) + n \end{aligned}$$

$$\therefore\ T(n) = n(n+1)/2$$

時間複雜度: $O(n^2)$

範例 2

- 若有一個時間函數 $T(n) = T(\sqrt{n}) + 1$,其中 $T(2) = 1$，求 $T(n) = ?$ Big-O =?

Sol:

$$T(n) = T(\sqrt{n}) + 1 = T(n^{1/2}) + 1$$

$$= [T(n^{1/4}) + 1] + 1 = T(n^{1/4}) + 2$$

$$= [T(n^{1/8}) + 1] + 2 = T(n^{1/8}) + 3$$

$$= \ldots$$

⇨ 令$n^{1/2^i} = 2$, 等號兩邊取$\log_2$, 則 $1/2^i \log_2 n = \log_2 2 = 1$

⇨ $\log_2 n = 2^i$, 等號兩邊再取$\log_2$ ⇨ $\log_2 \log_2 n = i$

$$= T(n^{1/2^i}) + i$$

(想辦法讓 $n^{1/2^i}$ 等於 2，以使T(2)=1)

$$= T(2) + \log_2 \log_2 n = 1 + \log_2 \log_2 n$$

⇨ $O(\log_2 \log_2 n)$

※範例練習※

- Example B.22

遞迴樹法 (Recursion-tree Method)

- 適用於母問題由**多個子問題**所構成
- 使用一個**樹狀結構**表示遞迴演算法則在執行過程被遞迴呼叫的情況，這個樹狀結構稱爲遞迴樹。其中:
  - **Node**: 存放遞迴關係式所相對應之子問題的Cost
  - **Degree**: 子問題的數目
- 遞迴樹法的三個步驟:
  - 按照遞迴方程式展開
  - 對**每一層的所有子問題**之cost加總，得到**每一層之cost**
  - 加總每一層的cost，以得到**total cost**，即爲答案
- 通常只能求出Big-O或Ω，若要計算 θ 得用“夾擠”法

範例 1

◆ 若遞迴式$T(n) = \underline{T(n/2) + T(n/4) + T(n/8)} + n$，試求$T(n) = \theta(n)$。(94成大)

◆ 三個不同的遞迴呼叫次數

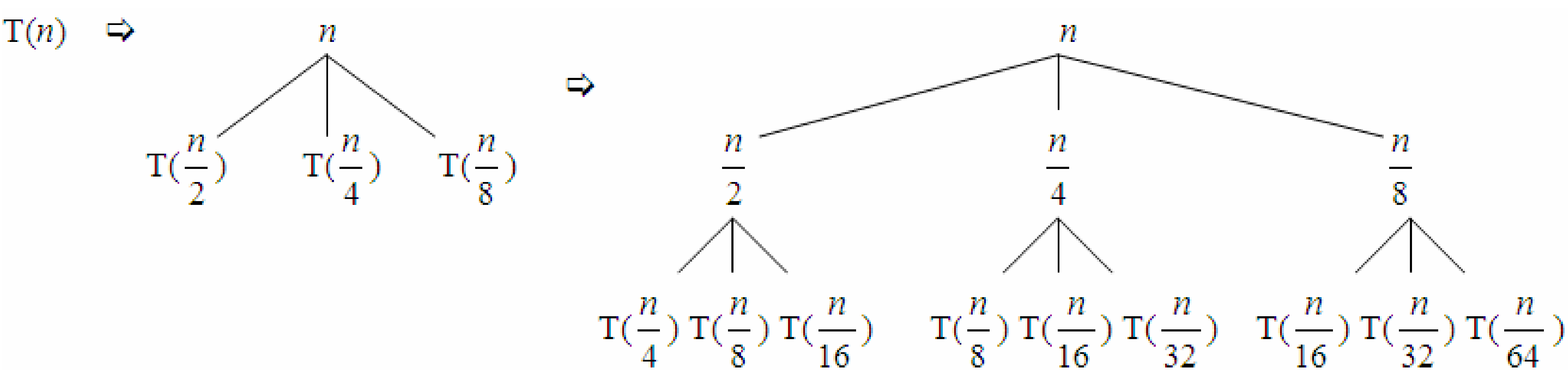
◆ 非遞迴工作的執行次數

Sol:

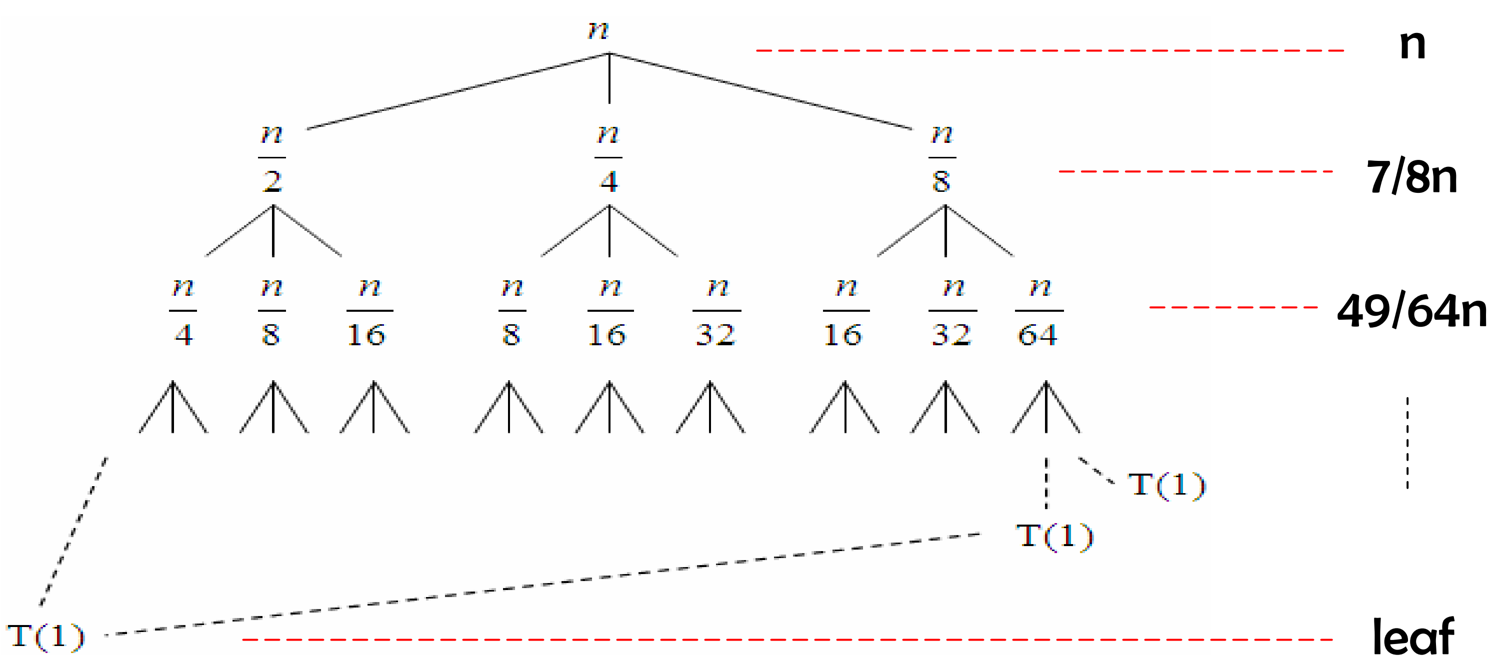
若要計算 θ 得用"夾擠法"，∴分別計算O和Ω

(1) 先求Big-O (即: 求Upper bound)

Step 1: (按照遞迴式展開)



Step 2: (計算每一層之cost)



Step 3: Total cost = n + 7/8n + 49/64n + … + leaf (公比小於1，用**無窮等比級數**求!!)

$\leq$ n + 7/8n + 49/64n + …

= n/(1-7/8)

$\therefore$ T(n) = **O(n)**

(2) 再求Ω (即: 求Lower bound)

(**Step 1**與**Step 2**同前，故不再求)

Step 3: Total cost = n + 7/8n + 49/64n + … + leaf

$\geq$ n

$\therefore$ T(n) = **Ω(n)**

$\therefore$ (1) + (2) ⇨ T(n) = **θ(n)**

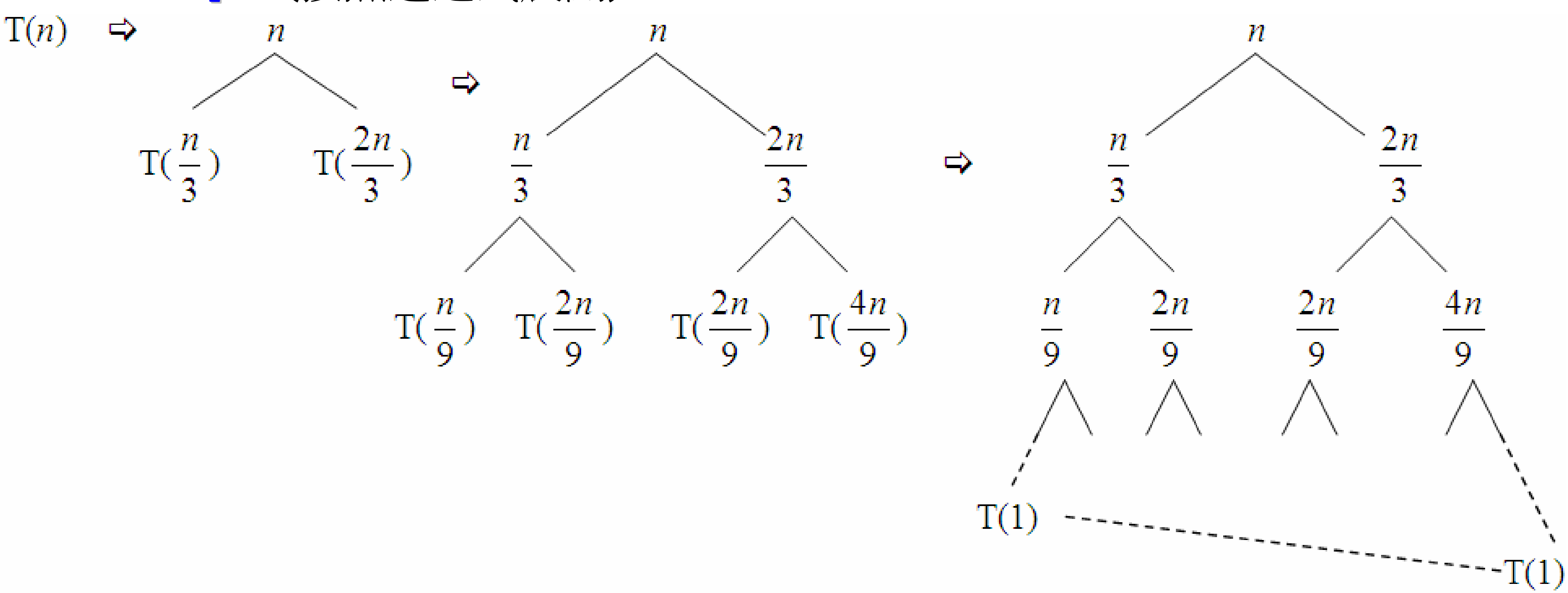
範例 2

- 若遞迴式T(n) = T(n/3) + T(2n/3) + n，試求T(n) = θ(?)。
  (90, 91 台大類似題)

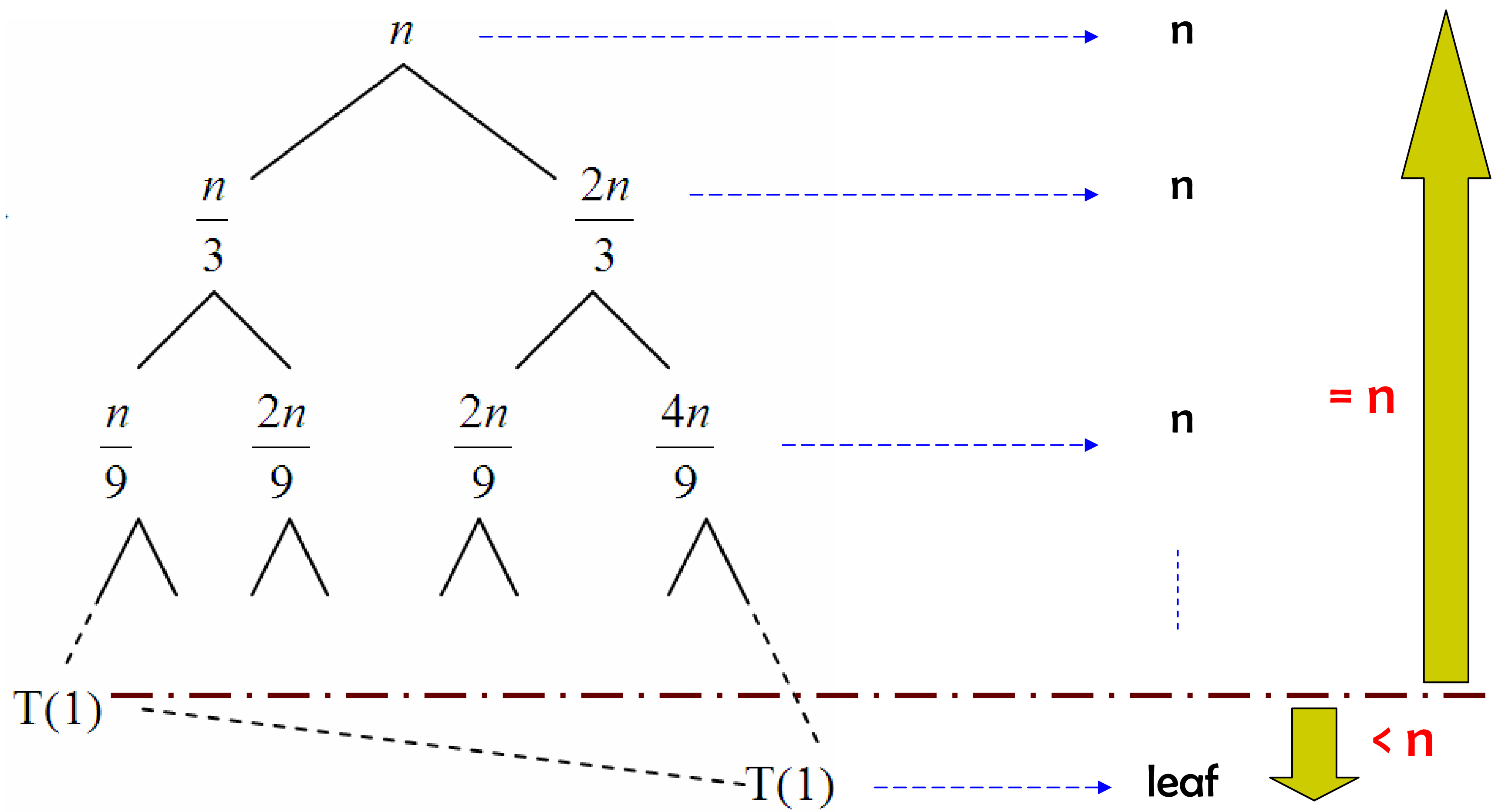
Sol: (若要計算 θ 得用“夾擠法”，∴分別計算O和Ω)

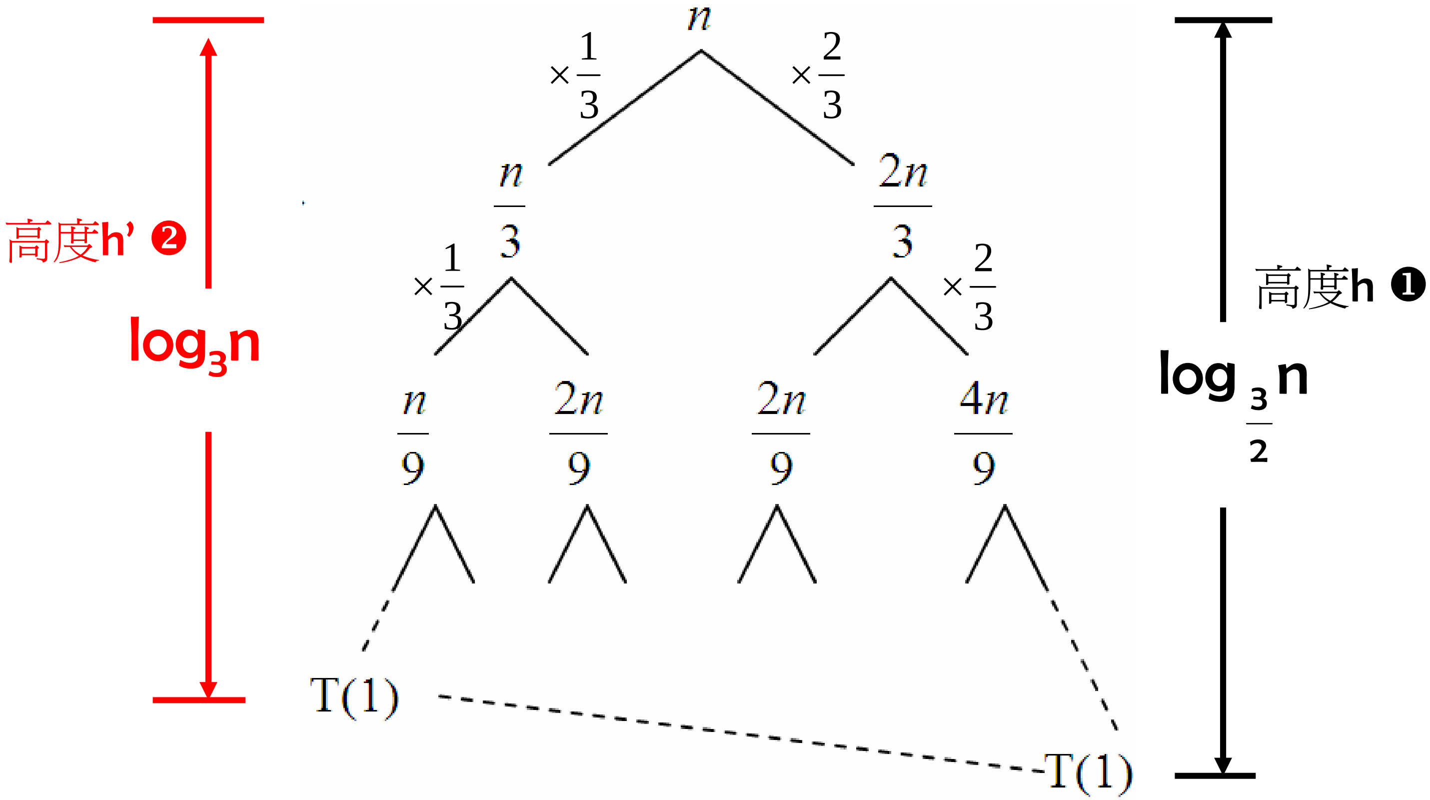
(1) 先求Big-O (即: 求Upper bound)

Step 1: (按照遞迴式展開)



Step 2: (計算每一層之cost)





分析 "高度h' ❷"

$n \rightarrow n/3 \rightarrow n/9 \rightarrow \ldots \rightarrow 1$

⇨ $n \times (1/3)^{h'} = 1,\quad 0 \leq h'$

⇨ $h' = \log_3 n$ ∴高度❷ = $\log_3 n + 1$

分析 "高度h ❶"

$n \rightarrow 2/3n \rightarrow 4/9n \rightarrow \ldots \rightarrow 1$

⇨ $n \times (2/3)^{h} = 1,\quad 0 \leq h$

⇨ $h = \log_{3/2} n$ ∴高度❶ = $\log_{3/2} n + 1$

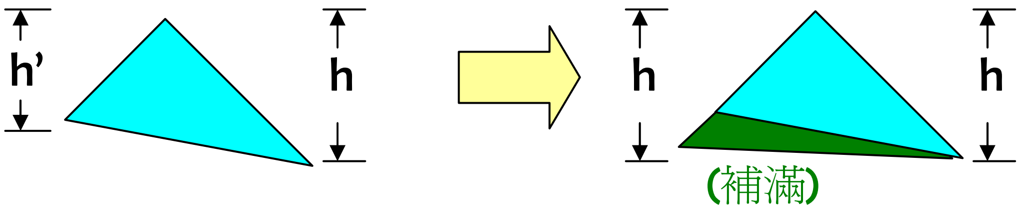
(1) 先求Big-O (即: 求**Upper bound**)

Step 3: Total cost = n + n + n + … + leaf

(公比 **r = 1**，不能用無窮等比級數來賴皮)

[解決方法]: 從 "高度❶" 下手

觀念:



$\therefore$ Total cost = n + n + n + … + leaf

$\leq$ n + n + n + … + n (有$\log_{3/2}n + 1$個n)

$= n(\log_{3/2}n+1)$

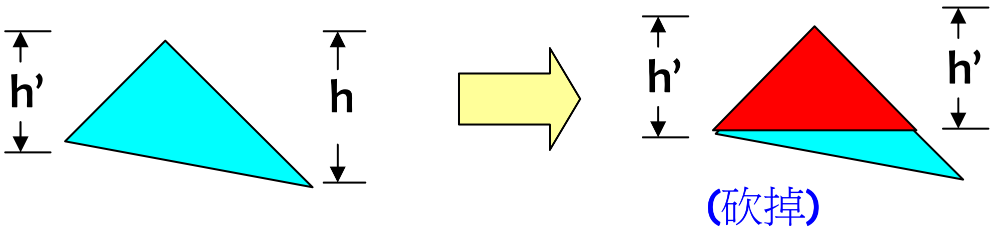
$\therefore$ T(n) = O(n log n)

(2) 再求Ω (即: 求**Lower bound**)

Step 3: Total cost = n + n + n + … + leaf

[解決方法]: 從 "高度❷" 下手

觀念:



∴ Total cost = n + n + n + … + leaf

$\geq$ n + n + n + … + n (有$\log_3 n$ + 1個n)

= n ($\log_3 n$+1)

∴ T(n) = Ω(n log n)

∴ 根據 (1)+(2)，得知 **T(n) = θ(n log n)**

※範例練習※

- 用遞迴樹法解$T(n) = T(n/5) + T(7n/10) + n$。
- 用遞迴樹法解$T(n) = T(n/2) + 2T(n/4) + n$。
- 用遞迴樹法解$T(n) = 3T(n/4) + cn^2$。

■支配理論方法 (Master Theorem Method)

- 為附錄B.5的Theorem B.5 與Theorem B.6 的擴展。
- 當遞迴方程式具有某種特定型式時適用。
- 【精神】讓 $f(n)$ 和 $n^{\log_b a}$ 比大小!!

【Master Theorem 支配理論】

令 $a \ge 1, b > 1$ 為兩常數，$f(n)$ 為一函數，時間函數 $T(n)$ 在非負整數下定義為 $T(n) = aT(\frac{n}{b}) + f(n)$，則:

① 若存在某個 $\varepsilon > 0$ 使得 $f(n) = O(n^{\log_b a - \varepsilon})$，則 $T(n) = \theta(n^{\log_b a})$。

② 若存在某個 $\varepsilon > 0$ 使得 $f(n) = \Omega(n^{\log_b a + \varepsilon})$，則 $T(n) = \theta(f(n))$。

③ 若 $f(n) = \theta(n^{\log_b a})$，則 $T(n) = \theta(n^{\log_b a} \times \lg n)$。

④ 若 $f(n) = \theta(n^{\log_b a} \times \lg^k n)$，其中 $k \ge 0$，則 $T(n) = \theta(n^{\log_b a} \times \lg^{k+1} n)$。

④ 是 ③ 的 General Case

範例 1

- 解 ❶. $T(n) = 8T(n/2) + n^2$, ❷. $T(n) = 4T(n/2) + n^2$

Sol:

❶. $a = 8, b = 2, f(n) = n^2$

⇨ $n^{\log_b a} = n^{\log_2 8} = n^3$

- ε = 0.5， ε 可自設，只要大於0 且 不違反後續條件 即可!!

$\because n^2 \leq n^{3-0.5}$

⇨ $f(n) = O(n^{\log_b a-\varepsilon})$ ... ①

$\therefore T(n) = \theta(n^{\log_b a}) = \theta(n^3)$

- 為何選用Big-O?
  - $\because n^{\log_2 8} = n^3$ 明顯大於 $f(n) = n^2$
  - Big-O 隱函有最大上限 (即 ≤) 之意
  - ∴選用Big-O表示

❷. $a = 4, b = 2, f(n) = n^2$

⇨ $n^{\log_b a} = n^{\log_2 4} = n^2$

$\because n^2 = n^2$ ⇨ $f(n) = \theta(n^{\log_b a})$ ... ③

$\therefore T(n) = \theta(n^{\log_b a} \times \lg n) = \theta(n^2 \lg n)$

- 為何選用Theta?
  - $\because n^{\log_2 4} = n^2$ 明顯等於 $f(n) = n^2$
  - Theta隱函有相等 (即 =) 之意
  - ∴選用Theta表示

範例 2

◆ 解 ❶. $T(n) = 3T(n/2) + n^2$, ❷. $T(n) = 4T(n/2) + n^2 \log n$

Sol:

❶. $a = 3, b = 2, f(n) = n^2$

⇨ $n^{\log_b a} = n^{\log_2 3} = n^{1.5\ldots}$

$\because n^2 \geq n^{1.5\ldots + 0.3}$

⇨ $f(n) = \Omega(n^{\log_b a + \varepsilon})$ … ②

$\therefore T(n) = \theta(f(n)) = \theta(n^2)$

❷. $a = 4, b = 2, f(n) = n^2 \log n, k = 1$

⇨ $n^{\log_b a} = n^{\log_2 4} = n^2$

⇨ 若 $f(n) = \theta(n^{\log_b a} \times \lg^k n)$, 則 $T(n) = \theta(n^{\log_b a} \times \lg^{k+1} n)$ … ④

$\therefore T(n) = \theta(n^{\log_b a} \times \lg^{k+1} n) = \theta(n^2 \lg^2 n)$

◆ 為何選用 Ω ?
- $\because n^{\log_2 3} = n^{1.5\ldots}$ 明顯小於 $f(n) = n^2$
- Ω 隱函有**最小下限** (即 ≥) 之意
- ∴選用 Ω 表示

◆ 為何選用Theta?
- $\because n^{\log_2 4} = n^2$ 明顯等於 n^2，而且 $n^{\log_2 4} \times \lg n = n^2 \lg n$ 也等於 $f(n) = n^2 \log n$
- Theta隱函有**相等** (即 =) 之意
- ∴選用Theta表示

※範例練習※

- 課本附錄B之Example B.26, Example B.27, Example B.28
- 解 $T(n) = 7T(n/2) + n^3$

  Ans: $\theta(n^3)$
- 解 $T(n) = 3T(n/2) + n^2\log n$

  Ans: $\theta(n^2 \lg n)$ (**Hint**: 本題不是以③或④來做判斷!!思考一下是以哪一個評判要件來決定的? 爲什麼？)
- 解 $T(n) = 4T(n/2) + n^2/\lg n$, $T(1) = c$

  Ans: $\theta(n^2 \lg \lg n)$ (**Hint**: 本題不能用支配定理，而是要用數學解法!!思考一下爲什麼？)

替代法 (Substitution Method)

- 即課本附錄B.1的歸納法 (Induction)之應用
- 適用於檢驗某個候選解答是否為此遞迴演算法的正確解，而不適用於求遞迴方程式的解答。
- 使用步驟:
  ① 利用猜測、觀察或匯整的方式，找出遞迴方程式的解 **(最難!!)**
  ② 利用數學歸納法証明此解是正確的
- 由於利用此方法求解遞迴方程式，最難的地方就是如何去猜出、觀察出或匯整出遞迴方程式的解。所以一般只適合當已有候選解時，用來驗証該解是否正確，也就是為了避開第一個步驟。

範例 1

- 請利用替代法找出 “Factorial” 的時間複雜度。Factorial的遞迴方程式 (時間函數) 如下:

$$T(n) = T(n-1) + 1$$

$$T(0) = 0$$

Sol:

① 利用猜測、觀察或匯整的方式，找出遞迴方程式的解

已知終止條件為T(0) = 0，我們可以嘗試匯整T(n)如下:

$$T(1) = T(1-1) + 1 = T(0) + 1 = 1$$

$$T(2) = T(2-1) + 1 = T(1) + 1 = 2$$

$$T(3) = T(3-1) + 1 = T(2) + 1 = 3$$

$$\dots$$

$$T(n) = n$$

② 利用**數學歸納法**証明 $T(n) = n$ 是正確的

數學歸納法的步驟:

a) 找出**歸納基底**: 對於n=0,

$$T(0) = 0 \qquad (\checkmark)$$

b) 做**歸納假設**: 假設對於任意正整數n，下列的式子成立:

$$T(n) = n$$

c) 找出**歸納步驟**: 我們必須証明

$$T(n+1) = n + 1$$

Proof:

$$T(n+1) = T(n+1-1) + 1 = T(n) +1 = n + 1 \qquad (\checkmark)$$

範例 2

- 請利用替代法找出下列遞迴方程式的時間複雜度。

$$T(n) = 2T(n/2) + n - 1, \quad \text{for } n > 1, n \text{ a power of } 2$$

$$T(1) = 0$$

Sol:

① 利用**猜測**、**觀察**或**匯整**的方式，找出遞迴方程式的解

已知終止條件爲T(1) = 0，我們嘗試匯整T(n)如下:

$$T(2) = 2T(2/2) + 2 - 1 = 2T(1) + 1 = 1$$

$$T(4) = 2T(4/2) + 4 - 1 = 2T(2) + 3 = 5$$

$$T(8) = 2T(8/2) + 8 - 1 = 2T(4) + 7 = 17$$

$$T(16) = 2T(16/2) + 16 - 1 = 2T(8) + 15 = 49$$

...

T(n) = **???** (難匯整)

※範例練習※

- 課本附錄B之Example B.1, Example B.2

補 充

補 1: 數學歸納法 (課本附錄A.3)

- 歸納法 (Induction) 是在當我們**已經有一個可能的推論結果**之後，用來**驗証**這個推論結果是否正確的工具。
- 它的步驟是：
  1) **驗証** n=1 時命題 f(n) 成立 (這叫**歸納的基礎**，或遞推的基礎)
  2) **假設** n=k 時命題f(n) 成立 (這叫**歸納假設**，或叫遞推的根據)
  3) **証明** n=k+1 於上述假設時，命題 f(n)成立。
- 為何需要數學歸納法
  - 只通過有限多個實例並不足以証明一個命題是成立的。那麼要如何証明一個命題對所有自然數都正確呢？

命題二： $f(n) = n^2 + \frac{(n-1)(n-2)\ldots(n-100)(\sqrt{2}-n^2)}{100!}$

若 n 是自然數則 $f(n) = n^2$

人們檢驗了 $n = 1, 2, 3, 4, \ldots, 100$ 發覺命題是成爲的即,

$f(1) = 1^2 = 1$

$f(2) = 2^2 = 4$

$f(3) = 3^2 = 9$

$f(4) = 4^2 = 16$

…

$f(100) = 100^2 = 10000$

但如此有規律的式子當 $n = 101$ 時你猜答案是多少?

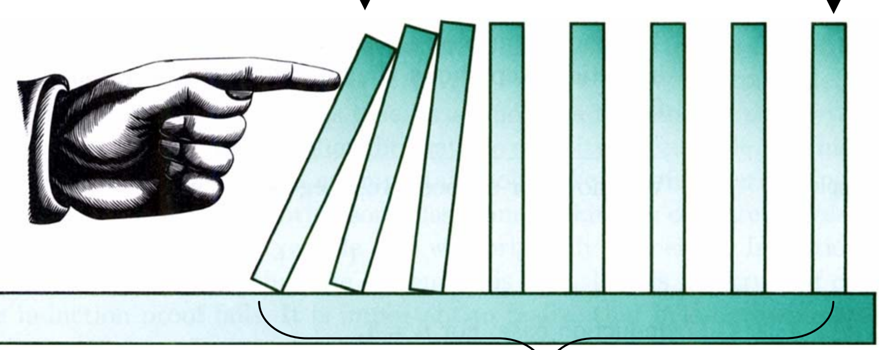
$f(101) = \sqrt{2}$

數學歸納法外一章

- 數學歸納法是說：有一批編了號碼的數學命題，我們能夠證明第 1 號命題是正確的；如果我們能夠證明當第 n 號命題正確時，則第 n+1 號命題也是正確的，那麼整批命題都是正確的了。
- 這是由於能夠證明第n 號命題是正確，並不能保證第n+1 號命題也會是正確的。名數學家華羅庚講過一個故事：
  - 「 一位買主買了一隻公雞回家。第一天，餵公雞一把米；第二天，又餵公雞一把米；第三天，還是餵公雞一把米。連續十天，每天都餵給公雞一把米。公雞就這十天的經驗，下了一個結論說：每天一定有一把米可吃。但是就在得出這個結論後不久，家裏來了一位客人，公雞就被宰殺成爲盤中飧請客人了。 」
- 華羅庚將這隻公雞如此得出結論的思考方法稱作公雞歸納法。而公雞歸納法是一種不完全歸納法。 華羅庚講這個故事的意思是說：「不能過分相信不完全歸納法。只對部分進行研究，得到一些結論，卻沒經過證明就說結論適用於全部，有時是要鬧出笑話的。 」
- 在數學發展史上這樣的例子不少，例如，法國數學家Legendre A.M在1798年研究過二次函數$f(n) = n^2+n+41$的值，當時他下了一個結論：它的函數值都是質數。而這個命題對不對呢？ 事實上，f(n) 經過計算，在$n \le 39$時，得到的值確實都是質數。但是這個命題還是錯的，因爲$n = 40$和41時， $f(40) = 40^2+40+41$ 及 $f(41) = 41^2+41+41$ 都不是質數。

- 歸納法的運作方式同骨牌效應 (Domino Principle)
  - 假設我們透過函數 f(n) 排列出下圖的骨牌陣，其中骨牌間距比骨牌高度小，則:
    - 我們可以推到第一張骨牌。
    - 假設只要任意兩相鄰骨牌的距離都比骨牌的高度小，我們就可以保証只要第n個骨牌倒下，第n+1個骨牌就會被推倒。

- 假設函數 f(n) 是正確的!!
  - ∵假設此函數正確，才有可能使任兩相鄰骨牌的距離比骨牌高度小
- 骨牌效應是以前後的結果來說明中間的未知情況



- 若我們根據某一個問題的一些情況，推論出某個函數f(n)，可利用歸納法來証明此函數是否為該問題的解。
  - 先証明當 **n = 1** (或某一起始值) 時，這個推論結果 (即: f(n)) 是成立的
  - 假設這個推論結果 f(n) 在n為任一正整數時是成立的，而當 **n = n+1** (或某一起始值) 也是成立的話，那麼這個歸納就是成立的。
- 歸納法 (Induction) 的証明步驟:
  - 找出歸納基底 (Induction base): 當 n=1 (或其它起始值)時，該推論結果為真的証明。
  - 做歸納假設 (Induction hypothesis): 對任一 n≥1 (或其它起始值)，假設該推論結果為真。
  - 找出歸納步驟 (Induction step): 當該推論結果對 n 為真，它對 n+1 也為真的証明。

- 試証對所有正整數n，下面的式子都成立:

$$1+2+\cdots+n=\frac{n\,(n+1)}{2}.$$

Sol:

1. 找出**歸納基底**: 對於**n=1,**

$$1=\frac{1\,(1+1)}{2}. \qquad (\checkmark)$$

2. 做**歸納假設**: 假設對於任意正整數**n**，下列的式子成立:

$$1+2+\cdots+n=\frac{n\,(n+1)}{2}.$$

3. 找出**歸納步驟**: 我們必須証明

$$1+2+\cdots+(n+1)=\frac{(n+1)\,[(n+1)+1]}{2}.$$

- 第3步的証明:

$$1+2+\cdots+(n+1) = \boxed{1+2+\cdots+n} + (n+1)$$
$$= \boxed{\frac{n(n+1)}{2}} + (n+1)$$
$$= \frac{n(n+1)+2(n+1)}{2}$$
$$= \frac{(n+1)(n+2)}{2}$$
$$= \frac{(n+1)[(n+1)+1]}{2}. \qquad (\checkmark)$$

- 由於我們<u>**已証明n=1時成立**</u>，根據骨牌效應，如果<u>**歸納假設是成立**</u>的，那麼<u>**在n+1的情況下也一定是成立的**</u>。因此，對所有的正數**n**都是成立。

補 1 相關例題

- Example A.2, Example A.3, Example A.4, Example A.5